Signature Router & Management System - Architecture Overview

Version: 1.0
Date: 2025-11-26
Status: Implementation Ready
Architect: BMAD Architect Agent

---

1. Executive Summary

Sistema bancario de enrutamiento y gestión de firmas digitales que orquesta múltiples canales (SMS, PUSH, VOICE, BIOMETRIC) optimizando costo, experiencia de usuario y resiliencia.

1.1 Características Clave

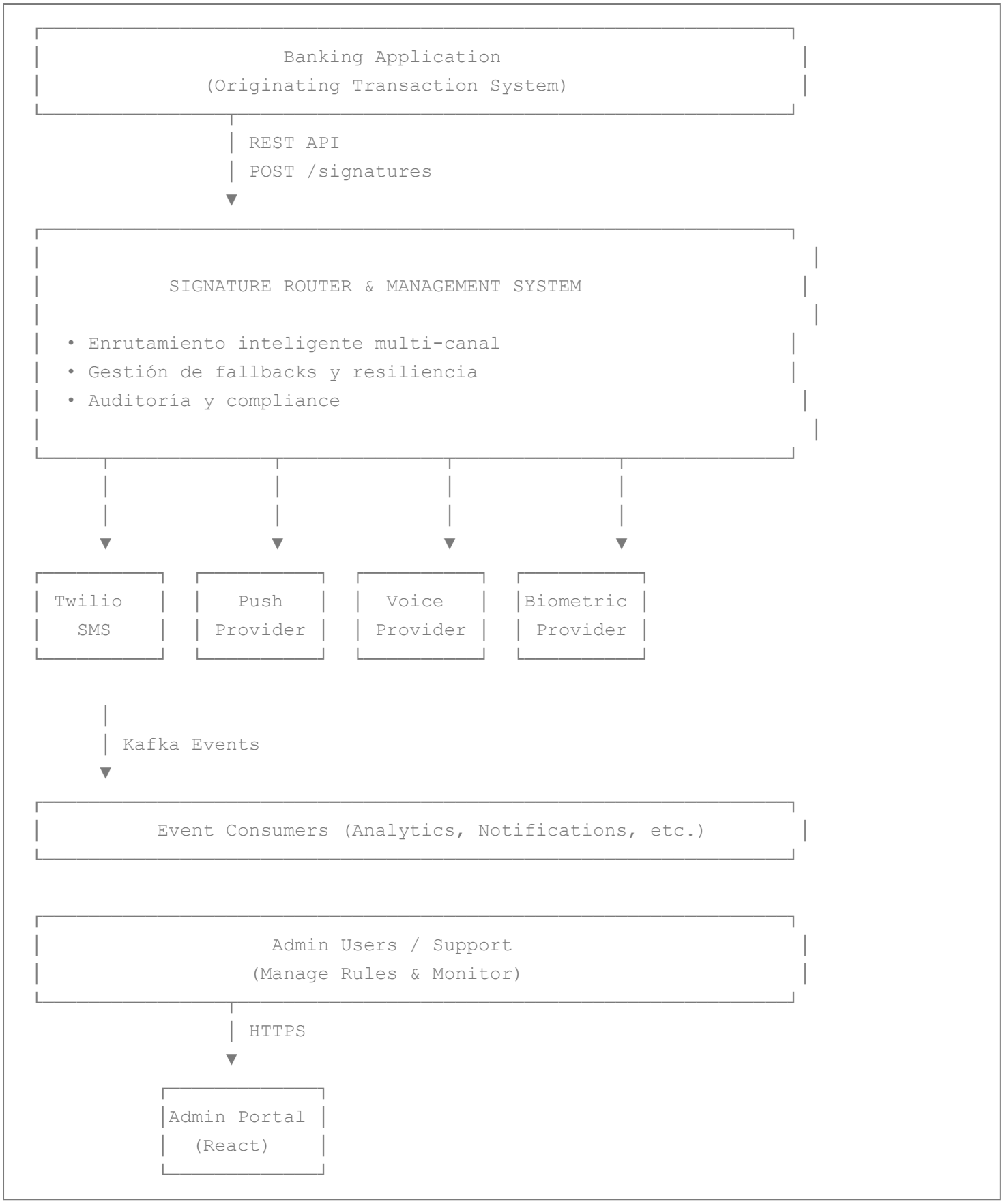
- **Enrutamiento Dinámico**: Evaluación de reglas SpEL con prioridades
- **Multi-Canal**: SMS (Twilio), Push, Voice, Biometrics
- **Fallback Automático**: Cadena de respaldo resiliente
- **Compliance Bancario**: PCI-DSS, GDPR, pseudonimización obligatoria
- **Event-Driven**: Kafka + Outbox pattern + Debezium CDC
- **Observabilidad Total**: Trazabilidad sin exponer PII
- **Admin Portal**: Gestión visual de reglas y análisis de costos

1.2 SLOs (Service Level Objectives)

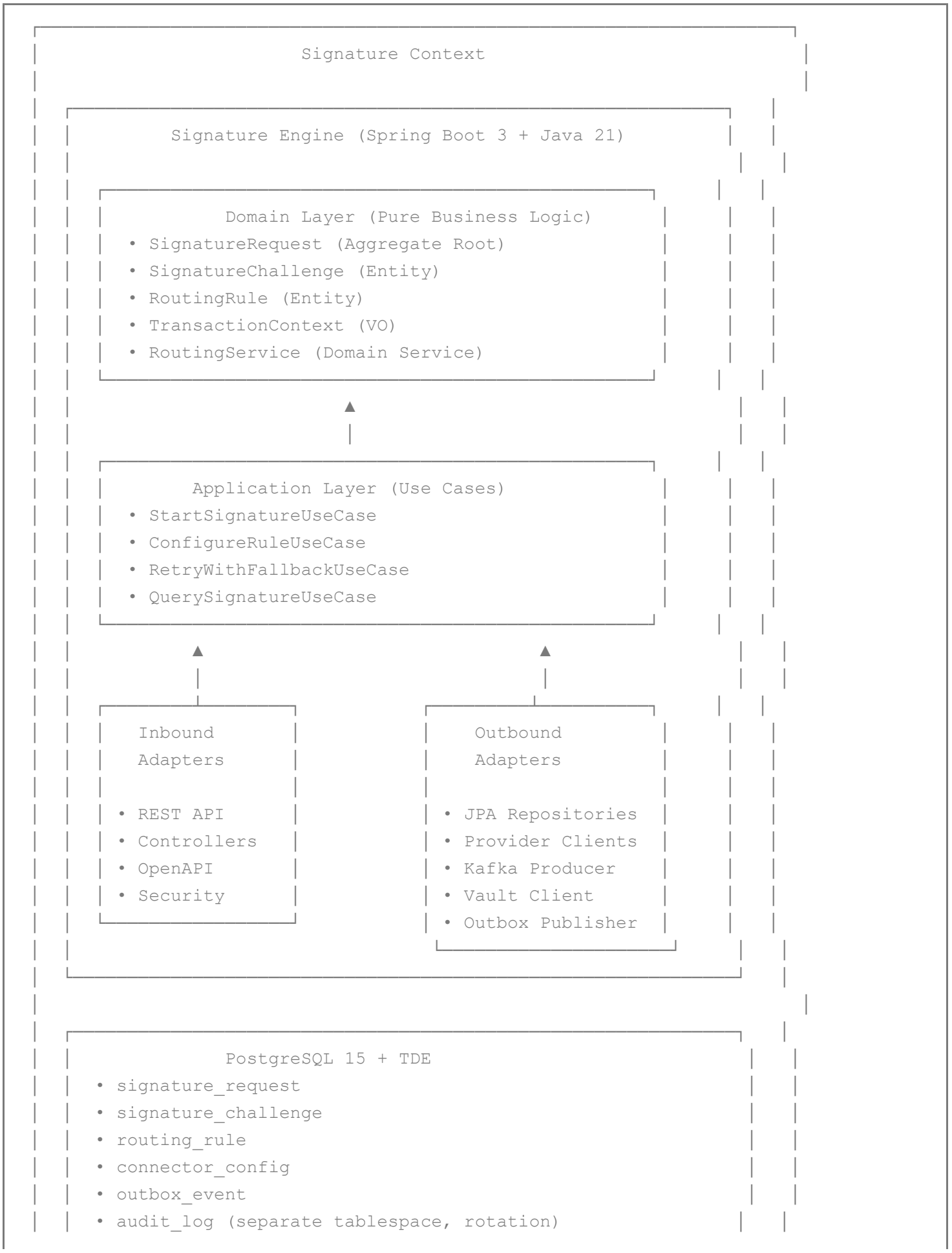
Métrica	Objetivo	Medición
Latencia P99	< 300ms	End-to-end signature request
Disponibilidad	≥ 99.9%	Uptime mensual
Error Rate	< 0.1%	Errores de negocio
Data Loss	0%	Event delivery guarantee

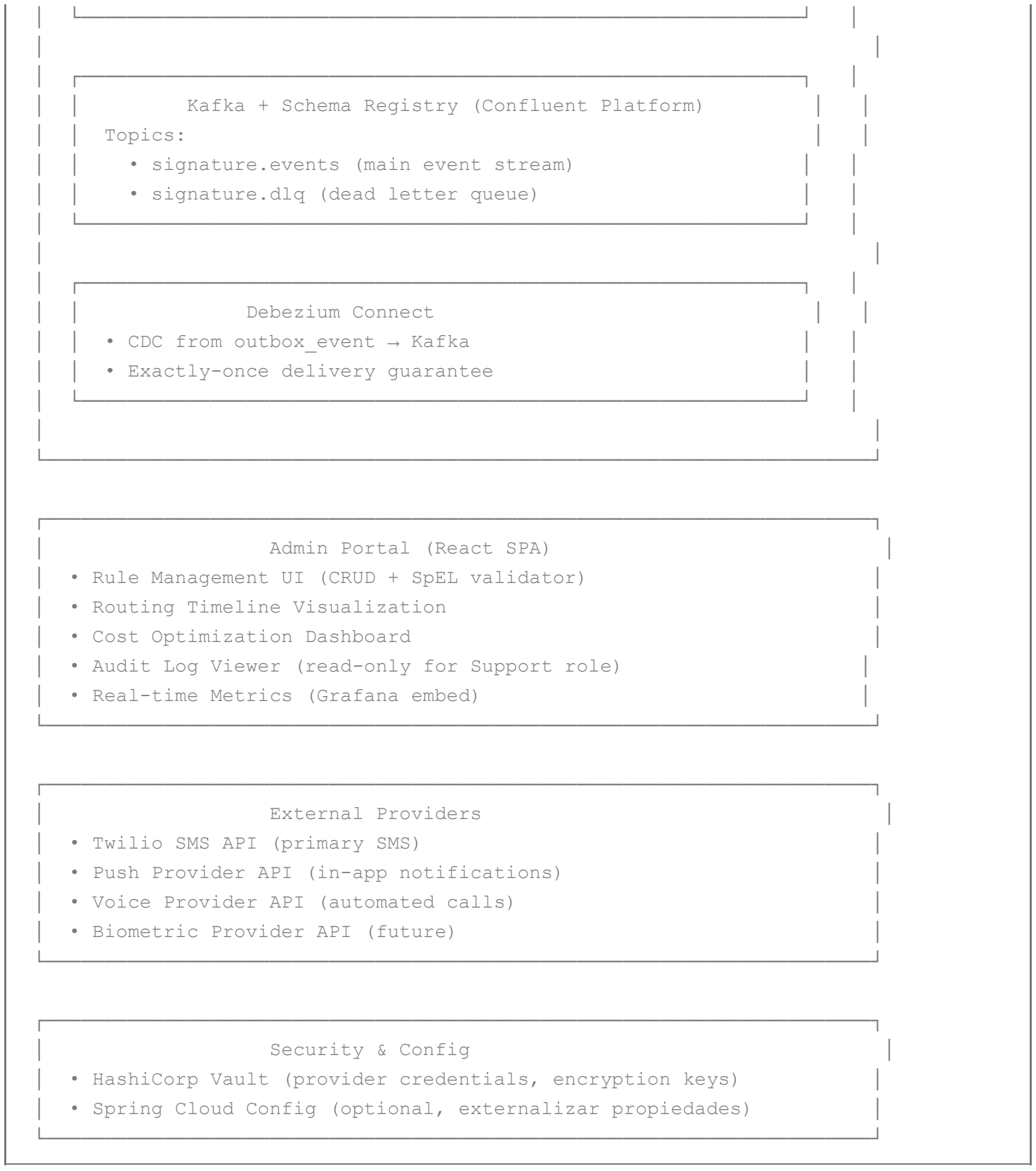
---

2. C4 Model - Context Diagram (Level 1)



3. C4 Container Diagram (Level 2)





---

4. Bounded Context: Signature Context

Este sistema contiene **UN solo bounded context** dado su alcance específico:

4.1 Ubiquitous Language

Término	Definición
SignatureRequest	Solicitud de firma digital para una transacción bancaria
SignatureChallenge	Desafío específico enviado por un canal (SMS, Push, etc.)
RoutingRule	Regla de negocio que determina el canal a utilizar
TransactionContext	Contexto inmutable de la transacción (JSONB)
Provider	Servicio externo que entrega desafíos (Twilio, etc.)
Fallback	Intento con un canal alternativo tras fallo
Degraded Mode	Estado donde un provider está temporalmente deshabilitado
Provider Proof	Recibo criptográfico del provider para no-repudio
Routing Timeline	Historial cronológico de intentos y resultados

4.2 Agregados y Entidades

```
SignatureRequest (Aggregate Root)
  └─ id: UUIDv7
  └─ customerId: String (pseudonimizado)
  └─ transactionContext: TransactionContext (JSONB inmutable)
  └─ status: SignatureStatus
  └─ activeChallengeId: UUID (nullable)
  └─ challenges: List<SignatureChallenge>
  └─ routingTimeline: List<RoutingEvent>
  └─ createdAt, updatedAt, expiresAt

SignatureChallenge (Entity dentro del agregado)
  └─ id: UUIDv7
  └─ signatureRequestId: UUID (FK)
  └─ channelType: ChannelType (SMS, PUSH, VOICE, BIOMETRIC)
  └─ provider: String (TWILIO, PUSH_SERVICE, etc.)
  └─ providerChallengeId: String
  └─ providerProof: String (recibo criptográfico)
  └─ status: ChallengeStatus
  └─ sentAt, respondedAt, expiresAt
  └─ errorCode: String (nullable)
  └─ rawResponse: String (nullable)
```

```
RoutingRule (Entity, gestión independiente)
  └─ id: UUIDv7
  └─ name: String
  └─ description: String
  └─ priority: Integer (menor = mayor prioridad)
  └─ condition: String (expresión SpEL)
  └─ targetChannel: ChannelType
  └─ enabled: Boolean
  └─ createdAt, updatedAt

ConnectorConfig (Entity)
  └─ id: UUIDv7
  └─ provider: String
  └─ enabled: Boolean
  └─ config: JSONB (timeout, retry, etc.)
  └─ vaultPath: String (ruta a credenciales)
```

---

5. Domain Services

5.1 RoutingService

```java
/**
 * Servicio de dominio que evalúa reglas de enrutamiento.
 * Responsabilidades:
 * - Evaluar condiciones SpEL contra TransactionContext
 * - Retornar el primer canal que coincida (short-circuit)
 * - No realiza llamadas externas (puro lógica de negocio)
 */
interface RoutingService {
    ChannelType evaluateRoute(
        TransactionContext context,
        List<RoutingRule> rules
    );
}
```

5.2 ChallengeService

```java
/**
 * Servicio de dominio para gestionar lifecycle de challenges.
 */
interface ChallengeService {
    SignatureChallenge createChallenge(
        SignatureRequest request,
        ChannelType channel
    );
```

```
    void markChallengeCompleted(SignatureChallenge challenge);
    void markChallengeFailed(SignatureChallenge challenge, String errorCode);

    boolean shouldAttemptFallback(SignatureRequest request);
    ChannelType determineFallbackChannel(ChannelType currentChannel);
}
```

---

6. Architecture Decisions

6.1 ¿Por qué Hexagonal Architecture?

- **Aislamiento del dominio**: Lógica de negocio bancaria crítica sin dependencias externas
- **Testabilidad**: Dominio testeable sin levantar infraestructura
- **Evolución**: Cambiar providers sin tocar lógica de negocio
- **Compliance**: Auditoría clara de cambios en reglas de negocio

6.2 ¿Por qué Outbox + Debezium?

- **Garantía de entrega**: Eventos nunca se pierden
- **Atomicidad**: Cambio de estado + evento en misma transacción DB
- **Desacoplamiento**: Sistema de firmas no depende de Kafka uptime
- **Orden garantizado**: Por aggregate root (signature_request_id)

6.3 ¿Por qué SpEL para Reglas?

- **Expresividad**: Condiciones complejas sin código custom
- **Validación**: Spring valida sintaxis antes de persistir
- **Seguridad**: Sandbox execution, no arbitrary code
- **Ejemplo**: `context.amount.value > 10000 && context.riskLevel == 'HIGH'`

6.4 ¿Por qué UUIDv7?

- **Sortable**: Clustering index eficiente en PostgreSQL
- **Distribución**: Generación sin coordinación
- **Timestamp embebido**: Debugging facilitado

---

7. Next Steps

Este documento es el fundamento para los siguientes artefactos:

1. ✅ System Overview (este documento)
2. ⏭️ Hexagonal Package Structure
3. ⏭️ Database Schema
4. ⏭️ Event Catalog
5. ⏭️ API Contracts (OpenAPI)
6. ⏭️ Resilience Strategy
7. ⏭️ Observability & Security
8. ⏭️ Admin Portal Architecture

Status: ✅ **COMPLETE - READY FOR REVIEW**